REMARKS

In the Office Action, the Examiner rejected the claims under 35 USC §102 and under 35 USC §103. The claims have been amended to further clarify the subject matter regarded as the invention. These objections and rejections are fully traversed below. Claims 1-25 remain pending.

Reconsideration of the application is respectfully requested based on the following remarks.

REJECTION OF CLAIMS UNDER 35 USC §102

In the Office Action, the Examiner rejected claims 1-7, 18-20, and 22-25 under 35 USC §102 as being anticipated by Crick et al, U.S. Patent No. 5,781,797, ('Crick' hereinafter). This rejection is fully traversed below.

Claim 1, as amended, relates to <u>DLLs</u>. Crick neither discloses nor suggests applying the claimed method to DLLs. In fact, in the Background section of Applicant's specification, various prior art methods of making library calls are addressed. As one example, "calling" and "returning" in a manner similar to subroutine calls is viewed as inefficient. Crick fails to address the problems associated with calling DLLs or a motivation to reduce the required execution time of DLLs. In addition, since DLLs are not loaded until they are needed, DLLs pose specific issues that do not exist with the component drivers in Crick. For instance, Crick indicates that the device drivers are configured at Computer startup (rather than when a specific device is being used). See., e.g., col. 5, lines 18-20.

Claim 1 as amended recites, in relevant part, "wherein each of the code modules responsible for calling a next one of the code modules in the chain includes a reference to the next one of the code modules in the chain, wherein an address in memory at which the next one of the code modules in the chain is loaded is associated with the reference to the next one of the code modules in the chain". Crick neither discloses nor suggests the invention of claim 1. Specifically, Crick neither discloses nor suggests including a reference to a next one of the code modules in the chain in a code module responsible for calling the next code module. In fact, Crick discloses a call-down table (e.g., stack) of a device driver. The component driver determines whether it should be inserted into the call-down table. See, e.g., col. 5, lines 18-47. Thus, a single call-down table is used, which would presumably result in various inefficiencies not present with the use of the invention of claim 1. Accordingly, Crick neither discloses nor suggests the invention of claim 1.

The above arguments equally applicable to independent claims 19, 22 and 25. Dependent claims 2-7, 18, 20, and 23-24 depend from one of the independent claims and are therefore patentable for at least the same reasons. However, the dependent claims recite additional limitations that further distinguish them from the cited references. Hence, it is submitted that the dependent claims are patentably distinct from Crick.

REJECTION OF CLAIMS UNDER 35 USC §103

In the Office Action, the Examiner rejected claims 8-9 and 21 under 35 USC §103 as being unpatentable over Crick. In addition, the Examiner rejected claims 10-17 under 35 USC §103 as being unpatentable over Crick in view of Pickett, U.S. Patent No. 6,374,400, ('Pickett' hereinafter). This rejection is fully traversed below.

Claims 8 and 9 have been cancelled, and the limitations have been incorporated into the independent claims.

As discussed above, Crick fails to disclose the presently claimed invention. Pickett fails to cure the deficiencies of the primary reference. In fact, FIG. 2 of Pickett neither discloses nor suggests a chain as claimed (or disclosed in Applicant's specification and illustrated in FIG. 1 of Applicant's specification), in which a calling DLL includes a reference to the next DLL to be called and an address of the next DLL to be called. Specifically, FIG. 2 illustrates a main code module, and illustrates that upon completion of

execution of a module, control is ultimately returned to the main code module. Moreover, FIG. 2 illustrates conventional subroutine calls as referred to in the Background section of Applicant's specification, and therefore teaches away from the claimed invention. This is referred to as a "typical program" in col. 4, lines 13-15 of Pickett.

The dependent claims recite additional limitations that further distinguish them from the cited references. For instance, as recited in claims 11-17, a branch table of a calling DLL is updated to include an address at which the called DLL is loaded. As another example, with respect to claim 15, two or more executable chains may be identified by a separate branch table entry and a separate address. Claim 16 depends from claim 15, and further recites that a parameter is associated with one of two or more executable chains such that each of the entries further includes a parameter used to select one of the two or more executable chains. Neither of the cited references, separately or in combination, discloses or suggests including a branch table in a calling DLL that is updated to include an address at which the called DLL is loaded. Moreover, neither of the cited references, separately or in combination, discloses or suggests accommodating different potential executable paths (i.e., chains) in different branch table entries as recited in claim 15, or distinguishing the executable chains by a parameter that is used to select one of the executable chains as recited in claim 16.

In addition, since DLLs are not loaded until the DLLs are called, the linking of the code modules presents problems that are not present with component drivers. Thus, as recited in claim 13, the branch table [is included in] of the first one of the one or more code modules includes the reference to the second one of the one or more code modules prior to loading the code modules and includes the address of the second one of the one or more code modules after the code modules have been loaded. In addition, with respect to claim 14, updating a branch table includes [creating] modifying an entry in the branch table such that a dummy address is replaced with [that identifies an entry point] the address of the second one of the one or more code modules. In other words, an address cannot be obtained until the DLL is loaded. At this time, the address is included in the branch table. Neither of the cited references, separately or in combination, discloses or suggests modifying a branch table entry to include an address at which a DLL has been loaded, or replacing a dummy address with an address at which a DLL has been loaded. Moreover, neither of the cited references discloses or suggests the necessity of including a reference to a DLL in a branch table entry and

updating the entry after the DLL has been loaded to include the load address. Accordingly, Applicant respectfully submits that the claims are patentable over the cited art.

Based on the foregoing, it is submitted that the claims are patentable over the cited references. The additional limitations recited in the independent claims or the dependent claims are not further discussed as the above discussed limitations are clearly sufficient to distinguish the claimed invention from the cited references. Thus, it is respectfully requested that the Examiner withdraw the rejection of claims under 35 USC §103.
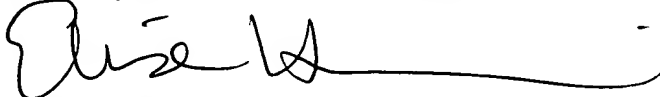
SUMMARY

If there are any issues remaining which the Examiner believes could be resolved through either a Supplemental Response or an Examiner's Amendment, the Examiner is respectfully requested to contact the undersigned attorney at the telephone number listed below.

Applicants hereby petition for an extension of time which may be required to maintain the pendency of this case, and any required fee for such extension or any further fee required in connection with the filing of this Amendment is to be charged to Deposit Account No. 50-0388 (Order No. CISCP125).

Respectfully submitted,
BEYER, WEAVER & THOMAS, LLP

Elise R. Heilbrunn
Reg. No. 42,649

BEYER, WEAVER & THOMAS, LLP
P.O. Box 778
Berkeley, California 94704
Tel. (510) 843-6200

APPENDIX – MARKED UP COPY OF THE CLAIMS

1.   (Once Amended)   A method of linking a set of code modules for execution, comprising:

determining one or more code modules to be executed, wherein the one or more code modules are one or more DLLs;

ascertaining a hierarchical order in which the one or more code modules are to be executed;

loading the one or more code modules to be executed; and

building a chain connecting the one or more code modules such that the one or more code modules will automatically execute in the hierarchical order when a first one of the one or more code modules is executed, wherein each of the code modules responsible for calling a next one of the code modules in the chain includes a reference to the next one of the code modules in the chain, wherein an address in memory at which the next one of the code modules in the chain is loaded is associated with the reference to the next one of the code modules in the chain.

2.   The method as recited in claim 1, wherein building a chain enables the one or more code modules to execute without requiring a parent code module responsible for calling the one or more code modules.

3.   The method as recited in claim 1, wherein the loading step is performed simultaneous with the building step.

4.   The method as recited in claim 1, wherein building a chain is performed such that the one or more code modules can be modified without requiring recompilation of the one or more code modules.

5.   The method as recited in claim 1, wherein loading the one or more code modules is performed in a reverse order of the hierarchical order.

6.   (Once Amended)   The method as recited in claim 1, wherein determining one or more code modules to be executed comprises determining one or more code·modules to be

executed to complete configuration of a hardware interface of a router.

7.   (Once Amended)   The method as recited in claim 1, wherein determining one or more code modules to be executed comprises determining one or more code modules to be executed to configure a [hardware device] router.

10.   (Once Amended)   The method as recited in claim 1, wherein building a chain connecting the one or more code modules comprises:

obtaining a first one of the one or more code modules, wherein the first one of the one or more code modules has previously been loaded;

determining whether the first one of the one or more code modules is to subsequently execute a second one of the one or more code modules upon completion of execution of the first one of the one or more code modules, wherein the second one of the one or more code modules has previously been loaded;

when it is determined that the first one of the one or more code modules is to subsequently execute a second one of the one or more code modules, updating a branch table [associated with] of the first one of the one or more code modules to identify an [entry point] address at which [of] the second one of the one or more code modules is loaded such that the reference to the second one of the one or more code modules in the branch table of the first one of the one or more code modules is associated with the address at which the second one of the one or more code modules is loaded.

11.   (Once Amended)   The method as recited in claim 1, wherein building a chain connecting the one or more code modules comprises:

obtaining a first one of the one or more code modules;

determining whether the first one of the one or more code modules has an option of executing a second one of the one or more code modules upon completion of execution of the first one of the one or more code modules;

when it is determined that the first one of the one or more code modules has an option of executing a second one of the one or more code modules, updating a branch table

[associated with] <u>in</u> the first one of the one or more code modules to identify an [entry point] <u>address at which</u> [of] the second one of the one or more code modules <u>is loaded such that the</u> <u>address of the second one of the one or more code modules is associated with the reference to</u> <u>the second one of the one or more code modules</u>.

12.   (Once Amended)   The method as recited in claim 1, wherein building a chain connecting the one or more code modules comprises:

obtaining a first one of the one or more code modules;

determining whether the first one of the one or more code modules can subsequently execute a second one of the one or more code modules upon completion of execution of the first one of the one or more code modules;

when it is determined that the first one of the one or more code modules can subsequently execute a second one of the one or more code modules, updating a branch table [associated with] <u>in</u> the first one of the one or more code modules to identify an [entry point] <u>address at which</u> [of] the second one of the one or more code modules <u>has been loaded such</u> <u>that the address of the second one of the one or more code modules is associated with the</u> <u>reference to the second one of the one or more code modules</u>.

13.   (Once Amended)   The method as recited in claim 12, wherein the branch table [is included in] <u>of</u> the first one of the one or more code modules <u>includes the reference to the</u> <u>second one of the one or more code modules prior to loading the code modules and includes</u> <u>the address of the second one of the one or more code modules after the code modules have</u> <u>been loaded</u>.

14.   (Once Amended)   The method as recited in claim 12, wherein updating a branch table includes [creating] <u>modifying</u> an entry in the branch table <u>such that a dummy address is</u> <u>replaced with</u> [that identifies an entry point] <u>the address</u> of the second one of the one or more code modules.

15.   (Once Amended)   The method as recited in claim 12, wherein when the first one of the one or more code modules is shared by two or more executable chains of code modules, associating the second one of the one or more code modules with one of the two or more executable chains <u>such that the branch table of the first one of the one or more code</u>

modules includes at least two entries, each of the entries identifying one of the two or more executable chains, each of the entries including an address.

16.   (Once Amended)   The method as recited in claim 15, wherein the second one of the one or more code modules is associated with one of the two or more executable chains when a parameter is associated with one of the two or more executable chains such that each of the entries further includes a parameter used to select one of the two or more executable chains.

17.   (Once Amended)   The method as recited in claim 12, wherein updating the branch table further includes replacing a dummy address associated with the reference to the second one of the one or more code modules with [an entry point] the address of the second one of the one or more code modules.

18.   The method as recited in claim 1, further comprising associating one of the one or more code modules with a hardware interface to identify a starting point for execution upon occurrence of an interrupt.

19.   (Once Amended)   A method of configuring a hardware device, the hardware device including a hardware interface, comprising

determining a configuration of the hardware interface, the configuration corresponding to one or more software modules to be executed to complete the configuration;

comparing the configuration against a set of rules that specify a hierarchical order in which the one or more software modules are to be executed in relation to one another;

identifying the one or more software modules that are to be executed;

ascertaining the hierarchical order in which the one or more software modules are to be executed;

loading the one or more software modules that are to be executed; and

building a chain connecting the one or more software modules such that the one or more software modules will automatically execute in the hierarchical order when a first one of the one or more software modules is executed, wherein each of the code modules responsible for calling a next one of the code modules in the chain includes a reference to the

next one of the code modules in the chain such that an address in memory at which the next one of the code modules in the chain is loaded is identified, the one or more software modules being one or more DLLs.

20.   The method as recited in claim 19, further comprising associating one of the one or more software modules with the hardware interface to identify a starting point for execution upon occurrence of an interrupt.

21.   The method as recited in claim 19, wherein the hardware device is a router.

22.   (Once Amended)   A computer-readable medium for linking a set of code modules for execution, the computer-readable medium storing computer-readable code, comprising:
        instructions for determining one or more code modules to be executed, the one or more code modules being one or more DLLs;
        instructions for ascertaining a hierarchical order in which the one or more code modules are to be executed;
        instructions for loading the one or more code modules to be executed; and
        instructions for building a chain connecting the one or more code modules such that the one or more code modules will automatically execute in the hierarchical order when a first one of the one or more code modules is executed, wherein each of the code modules responsible for calling a next one of the code modules in the chain includes a reference to the next one of the code modules in the chain, wherein an address in memory at which the next one of the code modules in the chain is loaded is associated with the reference to the next one of the code modules in the chain.

23.   The computer-readable medium as recited in claim 22, wherein the instructions for loading are to be executed simultaneous with the instructions for building the chain.

24.   The computer-readable medium as recited in claim 22, wherein the instructions for building a chain enables the one or more code modules to be modified without requiring recompilation of the one or more code modules.

25.   (Once Amended)   A system for linking a set of code modules for execution, comprising:

a processor; and

a memory, the memory storing therein:

instructions for determining one or more code modules to be executed, the one or more code modules being one or more DLLs;

instructions for ascertaining a hierarchical order in which the one or more code modules are to be executed;

instructions for loading the one or more code modules to be executed; and

instructions for building a chain connecting the one or more code modules such that the one or more code modules will automatically execute in the hierarchical order when a first one of the one or more code modules is executed, wherein each of the code modules responsible for calling a next one of the code modules in the chain includes a reference to the next one of the code modules in the chain, wherein an address in memory at which the next one of the code modules in the chain is loaded is associated with the reference to the next one of the code modules in the chain.